FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Serious Game for Learning Code Inspection Skills

Joaquim Pedro Ribeiro Guimarães

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Carlos Pascoal Faria

February 29, 2016

Serious Game for Learning Code Inspection Skills

Joaquim Pedro Ribeiro Guimarães

Mestrado Integrado em Engenharia Informática e Computação

February 29, 2016

Abstract

With the goal of turning software engineering learning more interesting and attractive, or more precisely code review techniques, an application was developed in 2013 that allows a teacher to make a set of challenges available that students may compete among themselves, finding the maximum number of errors in the least possible time. However this application contains some limitation in what concerns code reviews and inspections, since it only allows individual reviews, while the ideal review is usually done by a group. In addition, the application needs to be installed locally, which lowers both the spectrum of supported devices and accessibility.

The objective of this dissertation is to investigate, develop and implement an application that includes serious games concepts in code inspections in a way that allows for a better reproduction of a code inspection process. The main challenge is to reduce the limitations of the previous work, develop a feature that allows an approximate reproduction of what a code inspection is and add gamification concepts to it. To reach the solution, it was necessary to research gamification elements that favor the continuous use of the application and promote competitiveness.

This dissertation shows the research and investigation done, the implementation of the solution found, as well as the results of its testing.

Resumo

Com o objetivo de tornar mais interessante e aliciante o ensino de engenharia de software, ou mais exatamente de técnicas de revisão de código, foi desenvolvida em 2013 uma aplicação que permite a um professor disponibilizar um conjunto de desafios e onde os alunos podem competir entre si, descobrindo o maior número possível de erros no menor tempo possível. Contudo esta aplicação contém algumas limitações no que se refere a revisões e inspecções de código, pois apenas permite revisões individuais, enquanto que uma revisão ideal é realizada em grupo. Além disso, a aplicação requer instalação local, o que diminiu o espectro de dispositivos que a suportam e a facilidade de acesso à mesma.

O objectivo desta dissertação é investigar, desenvolver e implementar uma aplicação que inclui conceitos de jogos sérios em inspecções de código de modo a que seja permitida uma melhor reprodução de um processo de inspecção de código. O principal desafio é reduzir as limitações do trabalho anterior, desenvolver uma funcionalidade que permita uma reprodução aproximada do que é uma inspecção de código, ou seja, que permita revisões em grupo e adicionar conceitos de gamificação a essa funcionalidade. Para chegar à solução, foi necessário pesquisar elementos de gamificação que favorecem o uso contínuo da aplicação e que promovem competitividade.

Esta dissertação mostra a pesquisa e a investigação realizadas, a implementação da solução encontrada, assim como os resultados de testes efectuados.

Acknowledgements

This work wouldn't have been concluded without the help of the people that supported me through this project.

My sincere gratitude to my supervisor, João Carlos Pascoal Faria, for the challenge that was this project and the endless help provided, which was key for this dissertation to reach its conclusion.

My sincere gratitude to my course colleagues and work colleagues that helped me get through all the barriers that appeared in my way to the solution found and to the volunteers in the experiment (Bruno Lima, Mushtaq Raza, Inês Coimbra and Rui Pinto).

Finally, but not less important, my sincere gratitude to my family, that did everything tin their reach so I could achieve this goal and constantly support me through everything.

Thank you all.

Joaquim Pedro Ribeiro Guimarães

"On my business card, I am a corporate president.
In my mind, I am a game developer.
But in my heart, I am a gamer."

Satoru Iwata

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

API	Application Programming Interface
MVC	*Model-View-Controller*
FEUP	*Faculdade de Engenharia da Universidade do Porto*
SCOREL	*Source COde REview Learning*
IDE	Integrated Development Environment

Chapter 1

Introduction

1.1 Context and Motivation

Inspections and reviews consist in an important part of software development as they assure the produced artifacts meet the expected results and assure quality and efficiency. Enterprises make sure to integrate this step in the software development procedures and to dedicate specified staff responsible to assure inspections and reviews proceed according to the defined directives. However, it is a useful method for any programmer, as its benefits don't apply only to enterprise-based software, and it also helps programmers improve coding skills by helping create habits to better spot usual mistakes. Bearing this in mind, it is useful to improve code inspection skills and develop review habits, even though the current procedures are based on old fashioned methods like printed checklists. Since computers already help software be developed, it can be used to improve those procedures and make them more attractive to the regular programmer, as well as those who are starting to code. Besides creating a computer-based environment specific for those reviews, it would be interesting to apply *gamification* elements and create serious games to help turn those procedures in something more captivating to the user.

The usual method of code inspection is unappealing, as it requires documents exchange or inadequate platforms and tools to execute the different tasks it requires, if not done in the presence of everyone. In case the inspection is not done by a group but rather individually, it's hard to do so autonomously since it loses the advantages of the group revision. In what concerns learning, it's hard to evaluate automatically, as the solution is required to be available to everyone, and even after that, it may exist human error in comparing the student's revision to the solution.

With that in mind, in 2013 a FEUP student by the name of Edgar Alves developed SCOREL, an application in which students could solve code inspection exercises posted by teachers. However, only the individual revision can be done using this application, and it requires to be installed in the computer, which are limitations that can see improvements.

1.2 Objectives and Expected Results

The main goal is to apply gamification techniques to turn code revision and inspection learning into a more appealing, effective and efficient process, instead of the classic methods, which can be tiring and not so effective, since usually it requires the presence of the participants or the use of inadequate platforms. In this way, the final product shall be a web based game that can simulate the code inspection process, in a group level, and code revision exercises, in an individual level.

1.3 Document Structure

After the Introduction, this dissertation contains 4 more chapters. In chapter 2, the state of the art is described, as well as previous work, with special focus on serious games, gamification, software reviews and inspections and the work done in the field by Edgar Alves. In chapter 3, are presented the proposed solutions and contributions, detailed by user stories, views and features. In chapter 4, the validation experiment is described and its results analysed. In chapter 5, is described the possible implementation of the solution, detailed by user stories, views and features. In chapter 6 there is the conclusion to this document.

Chapter 2

State of the Art

In this chapter the state of the art of the areas related to the subject of this dissertation is described.

2.1 Serious Games and Gamification

A game is a form of interactive entertainment where players must overcome challenges, by taking actions that are governed by rules, in order to meet a victory condition, according to Rollings and Adams on Game Design.

We can define a serious game as a digital game designed with the purpose of solving a problem, not just for entertainment, being that problem derived from subjects such as training, research, advertising and marketing. [Coe]

The main elements of a game are the mechanics, which are the procedures and rules of the game, the story, which is the sequence of events that unfold in a game, the aesthetics, which represents how the game looks and feels, and the technology, which is the medium through which we access the game.

The human brain is capable of learning or play to create stories. This capacity allows the act of learning to be amusing, which will create more interest and motivation in the act of learning. Furthermore, Johan Huizinga[Hui38] considers the game as something innate to humans and even animals, considering it an absolutely primary category of life, prior to culture.

Since the 1980 decade, games have been used as an education tool and for training, to explore the factor that games increase motivation and interest. Besides that factor, there are others that can make a strong defense for the use of games in learning activities, such as the ability of games to provide a platform for active learning, or in other words, to do something instead of just listening and reading, possible costumizations, the ability to provide immediate feedback, to allow active discovery and develop new kinds of comprehension and reach an higher level of retention of material.

There is a concept to describe situations such as learning being turned into a game, which goes by the name of gamification an can be defined as the use of game design elements and techniques in non-game contexts. Gamification is a part of the User Experience Design. [Coe15]

To apply gamification to a situation, the participant or main actor in that situation should be considered the player and be considered the center of the game with a sense of control. The goal is to make the player play and keep playing. The process of making the player start the game is to give that player an intuitive progression and provide mastery. Players should perceive the game as consistent, fair and fun, usually by:

- Consistent challenges
- Perceivably fair playing experiences
- Lack of stagnation
- Lack of trivial decisions
- Difficulty levels

The focus should be to create an experience for the player. Since it has to be fun, usually, according to Marc LeBlanc, a game designer, game pleasures come from the following topics: [HLZ]

- Sensation: Using your senses to sense the game world
- Fantasy: Imaginary world
- Narrative: The dramatic unfolding of a sequence of events
- Challenge: One of the core pleasures, related to problem solving
- Fellowship: Cooperation and communities
- Discovery: Exploring the game world and discovering secret features
- Expression: The player expresses himself by creating something
- Submission: The suspension of disbelief

As for fun, Nicole Lazzaro defines fours keys to fun: easy fun, hard fun, people fun and serious fun. [Laz04]

Game elements can be divided in 3 categories: dynamics, mechanics and components. The dynamics are constraints, emotions, the narrative, progression and relationships. The mechanics consist in challenges, chance, competition, cooperation, feedback, resource acquisition, rewards, transactions, turns and win states. Finally the components are achievements, avatars, badges, boss fights, collections, combat features, content unlocking, gifting, leaderboards, levels, points, quests, social graphs, teams and virtual goods.

A good way of motivating the player is using the PBL triad: points, badges and leaderboards. Although these elements produce great results in what regards player motivation and satisfaction, these elements are not the game itself and they may not be fun to achieve, which can result in the opposite effect.

Behaviorism is present in gamification. The Cognitive Evaluation Theory[Pi] defines a reward typology according to the following topics:

- If they are tangible or virtual;
- If they are expected or a surprise, which is also a game pleasure;
- What the players have to do to get the reward, being it by the beginning or completion of a task, by performance or for no reason at all.

However, behaviorism has limits, since players can decide that the rewards are not what they want or not their focus or enter the Hedonic treadmill, which means that once rewarding becomes focus it has to be maintained because it does not change the behavior.

Werbach's six-step Gamification Design Framework[Wer13] consists in the following:

- Define business objectives: consists in listing and ranking possible objectives, eliminating means to ends and justify objectives.
- Delineate target behaviors: such behaviors must be specific and based on metrics and analytics, allowing players to receive feedback on their attempts to engage in the intended behavior.
- Describe players: can be done using demographics, psychographics, Bartle's player types [Bar96] or any other framework.
- Devise activity loops: should describe in detail the plan to motivate players using engagement and progression loops.
- Remember the fun: although more abstract, it is as important as the other items.
- Deploy the appropriate tools: consists in considering the game elements and what will be their experience to players.

However, gamification also holds criticism and risks, from which exploitationware, cheating, legal issues and regulatory issues can be highlighted, since they can bring down the whole environment behind a game. [Per14] [UW10]

2.2 Good Practices of Code Reviews and Inspections

Reviews are an important part of software development since they improve software and take it closer to the expectations. In reviews it's meant that one or more peers have inspected or evaluated a software artifact.

By definition, a review is usually a group meeting whose purpose is to evaluate one or more software artifacts and the general goals are to identify problems, errors and defects early in the software life cycle, ensure the artifact conforms to organizational standards and identify components that do not need improvement. Its benefits stand in software improving in quality, increasing productivity by shortening the rework time, closer adherence to project schedules and increased awareness of quality issues. [Wie01]

Reviews have the advantage of a two-step approach, where the first step is to read the reviewed item individually and, the second step, by the group as a whole. Since developers usually work alone, they may spend hours trying to locate a defect that will be quickly detected in a group. Reviews policies should specify when reviews take place, what is to be reviewed, types of reviews that will take place, who is responsible, what training is required and what review deliverables are.

Reviews can be formal or informal and technical or managerial. They can be as simple as asking for a peer or colleague to review code or as complex as a group meeting of individuals not part of the coding staff, that produces a document delivered to the programmers of the artifact in question. The first, more used in a less formal environments due to its simplicity, is more useful when there aren't many resources that can be spent on reviewing - like the number of available colleagues - or the artifact being reviewed isn't very extensive. However, it's still effective in what concerns defect detection and returning feedback to the programmer. The second will be explained in more detail further in this chapter.

Managerial reviews usually focus project management status, while technical ones are used to verify that a software artifact meets its specification, to detect defects and check for compliance with standards. Informal reviews can be used as a form for colleagues to communicate and get peer input on their work while formal reviews require written reports to summarize the process.

Inside technical reviews we can find two types: inspections and walkthroughs. Inspections are a type of review that is formal in nature, which requires a group usually composed by 3 to 7 members but very depending on the item size and the experience of the participants, with a leader or moderator, usually a member of the technical staff or the quality assurance team, preferably unrelated to the project to preserve objectivity. The leader schedules and plans the inspection, appoints someone to record results, runs the process and monitors the aftermath of the review. As part of planning, sets a checklist that varies with the software artifact being reviewed, containing items that inspection participants should focus and evaluate, being the completed checklist part of the review summary document. Checklists should hold fields for the identification of the problem or defect type, status, severity grade and a field to indicate if it's missing, incorrect or superfluous. However, they can also consist on mere topics to be checked if the software verifies the condition in each topic. An example of checklist can be found here [Cre]. The inspection starts when the inspection preconditions are met, after which the leader distributes the items to be inspected along with the checklist. During the reviewing process problems and errors should be noted for discussion in the meeting following the review. When that meeting takes place the produced review document is presented and discussed, with focus in quality, adherence to standards, testability, traceability and satisfaction of the users/clients requirements. Inspection metrics are also recorded.

In the end, the leader signs a summary report. Usually there's a follow-up process to ensure the issues identified in the inspection are addressed.

Walkthroughs are another type of technical review where the producer of the reviewed material serves as review leader and guides the progression of the review. In case of design or code walkthroughs, test inputs may be selected. It can be used for material other than code, like data descriptions, reference manuals or specifications. This type of review eliminates the need for a checklist.

It is possible to understand that a review is a very well planned process. In that way, there should be a certain care with review plans. A template can be applied and should specify:

- Review goals
- Items being reviewed
- Preconditions for the review
- Roles, team size and participants
- Training requirements
- Review steps
- Checklists and other documents to be disturbed to participants
- Time requirements
- The nature of the review log and summary report
- Rework and follow-up

In what regards review goals, we can define the following:

- Identification of problematic components or components in the software that need improvement
- Identification of specific errors or defects in the software artifact
- Ensuring that the artifact conforms to organizational standards
- Communication to the staff about the nature of the product being developed

Examples of items that can be reviewed are:

- Requirements documents
- Design documents
- Code

- Test plans
- User manuals
- Training manuals
- Standard documents

It is useful to quantify and qualify the work done in a review and for that purpose there are specific metrics. The most relevant metrics in reviews are the following:

- Size of item reviewed
- Review time
- Number of defects found
- Number of defects that have escaped and were found in later review and testing activities, and finally in operation by the user
- Review yeld, which is the percentage of existing problems that were detected
- The number of defects found per hour of reviews time, also known as Defect Removal Rate
- Number of defects found per page or per line of code (or KLOC if the number of lines is measured in the thousands), which can be named Defect Density
- Lines of code or pages of document that were reviewed per hour, or Review Rate
- Defect removal leverage

$$DRL = \frac{Defects\ per\ hour\ in\ review\ or\ test\ phase\ X}{Defects\ per\ hour\ in\ review\ or\ test\ phase\ Y} \tag{2.1}$$

[Bur02]

2.3 Computer-based Learning for Software Reviews Education

Gamification in software engineering learning is not a new concept. There's a range of games being used to teach software development and increase related skills already in use and with results, in such a way that it was adopted as an established method in companies of the industry, as it is possible to observe in Table 2.1.

Table 2.1: Examples of gamification application

[US14]

Company	**Area**	**Game**
IBM	Corporate social collaboration	IBM Connections
IBM	Business process management	IBM Innov8
IBM	Reduce of cost of internal business	Document Translation
Microsoft	Improvement of productivity/quality assurance	Windows Language Quality Game
Microsoft	Gamification of threat assessment process	Elevation of Privilege
HP	User engagement	HP Operations Manager
HP	Increase of sales	Project Everest
HP	Improvement of conference paper's selection	HP global technical conference
Oracle	Employee on-boarding	New Hire
Cisco	Building a team	The Threshold
Cisco	Improvement of global sales experience	The Hunt
Siemens	Training and development	Plantville
PWC	Employee recruitment	Multipoly
SAP	Corporate social network (car-pooling)	Two-Go
SAL	Sales training	SAP Road Warrior
SAP	Marketing and branding	Paul the Octopus
Google	Employee recruitment	Google code jam
NTTDate	Leadership training	Ignite Samurai Leadership
Accenture	Employee flexibility	Liquid Workforce
American Express	Business travel	Global Business Travel

[US14]

In what concerns serious games in code inspections and reviews, the amount of established work is reduced to one application, called SCOREL and developed by Edgar Alves, a FEUP alumni, for his dissertation.

Figure 2.1: SCOREL playing environment

SCOREL consists in an application developed in Java, which allowed a professor to upload to a server RTF and HTML files with source code and mark chunks with specific types of errors so that a student, via a list of exercises registered in the server, can access an exercise and solve it, matching his answer with the list the professor made at the time of the exercise upload. [Alv13]

Although the application met its goals, it holds some limitations as it doesn't allow code inspections in their definition, as it just simulates an individual review, and the ideal inspection environment prefers a group to execute individual reviews and exchange results, also lacking some form of difficulty levels to help a gradual learning environment and adaptable metric evaluation in that regard. The fact that it is an application the user needs to install, although not troublesome, is still a point where it shows room for improvements.

2.4 Integration in Current Work

There isn't an established serious game that covers the subject of code inspections. However, the state of the art described in this chapter indicates it's possible to create such a game, based on work already investigated and developed.

Starting with the work developed by Edgar Alves, its game format can be the basis of the game modes of the new application. The games will be time based and the goal will also be to find defects in a determined period of time. However, and according to what was researched about code inspections, defects can have various classifications. So, that will also have to be considered, as well as the implementation of *gamification* concepts.

Chapter 3

Solution Conception

In this chapter is described the elements of the conception of the solution found, based on the goals set.

3.1 User Stories

This application allows a type of user to post exercises and another type to solve them, individually or in a team. With that in mind, there are two actors: the teacher - the one who posts the challenges - and the student - the one who solves them.

The user stories are divided in 3 types according to the actor: teacher, student and available to both. This separation is identified via the first digit in the identifier, being 0 for the general user stories, 1 for teacher related user stories and 2 for those where the actor is the student only.

Table 3.1: US01 - Login

Identifier	*US03*
Description	*Login*
Actor	*Teacher and Student*
Precondition	*Have a registration (intended to work with the database of the institution)*
Process	*Enter credentials in the login area*
Priority	*High*
Effort	*1*

Table 3.2: US02 - Logout

Identifier	*US02*
Description	*Logout*
Actor	*Teacher and Student*
Precondition	*Be logged in*
Process	*Press the logout button*
Priority	*High*
Effort	*1*

Table 3.3: US11 - Create a challenge

Identifier	*US11*
Description	*Create a challenge*
Actor	*Teacher*
Precondition	*Logged as teacher*
Process	*Access "New challenge" in the dashboard and choose a file to upload, then choosing the settings*
Priority	*Medium*
Effort	*3*

Table 3.4: US12 - Verify challenge results

Identifier	*US12*
Description	*Verify challenge results*
Actor	*Teacher*
Precondition	*Logged as teacher*
Process	*Access the challenge and enter the statistics area*
Priority	*High*
Effort	*5*

Table 3.5: US13 - Consult student results

Identifier	*US13*
Description	*Consult student results*
Actor	*Teacher*
Precondition	*Logged as teacher*
Process	*Access the list of students of the course edition wanted and select the student*
Priority	*Medium*
Effort	*5*

Table 3.6: US21 - Single player challenge

Identifier	*US21*
Description	*Single player challenge*
Actor	*Student*
Precondition	*Logged as student*
Process	*Access open challenges area in dashboard or access course page and confirm start. When the challenge starts, proceed to signal errors in code reviews. When the challenge is over, visualize result and exit.*
Priority	*High*
Effort	*7*

Table 3.7: US22 - Multi player challenge

Identifier	*US22*
Description	*Multi player challenge*
Actor	*Student*
Precondition	*Logged as student*
Process	*Access open challenges area in dashboard or access course page and join a team, pressing a button signaling readiness. When the challenge starts, proceed to signal errors in code reviews. When the challenge is over, agree final answer with the rest of the team and submit. Then, visualize result and exit.*
Priority	*High*
Effort	*11*

Table 3.8: US23 - Consult own results, badges and achievements

Identifier	*US23*
Description	*Consult own results*
Actor	*Student*
Precondition	*Logged as student*
Process	*Access statistics area in personal dashboard*
Priority	*Medium*
Effort	*5*

3.2 Features

This section lists the features of the designed solution.

3.2.1 Pinpoint defects

In the game view, the goal is to mark defects, and to do that the player has to follow some steps.

The player has to be sure that there is some code selected in the code area, that there is a selected type and one of the two severity options is also selected. Once all those conditions are met, the player can click the "Pin" button to mark the excerpt as a defect with the intended severity and type. In the code, the excerpt will be highlighted and the defect will be added to the list in the sidebar, having a border with a certain color according to the selected severity and the description equal to the selected type. From that point, the player can either click on the defect on the list to jump to the defect, in which the code area, if longer than the screen height, will scroll to the marking position and turn the highlight color orange, in order to show the selected defect among the others, or unmarking the defect by clicking at the rightmost part of its entry in the defect list.

3.2.2 Vote up or down defects

In the meeting view, the players without the captain role will have to help the captain by voting up or down the defects according to their opinion on whether the defect should be submitted in the solution attempt. To do so, they click on one of the displayed numbers in the intended defect, on the defect list. The number to the left represents the votes in favor and will be green if the defect is voted up by the user. The same happens with the number to the right, only it turns red instead. It's possible to undo a vote, but it's not possible to vote up and down on the same defect. Voting up will undo a vote down, if the player voted that defect down before and vice voting down will undo a vote up, if the player voted that defect up before.

3.2.3 Pick and unpick defects

Also in the meeting view, but for the players with the captain role, there are a specific set of actions designed to define the defects that will be part of the team's attempt. The captain simply picks or unpicks a defect according to the available option for the defect, that depends on its current state. If the defect is picked, it will be colored on the list and marked on the code in a bright color, with the available option being to unpick. If it's unpicked, it will be gray on the list and marked with a darker color on the code, while the available option is to pick the defect.

3.2.4 Creating or joining a team

When accessing the team game, the players enter the team lobby for the selected exercise. In that screen, each player can create a team by clicking the respective button or join a team that already exists by clicking on its title.

The players then, just have to click to signal they are ready and wait for the captain selection and to be redirected to the game page.

3.3 Game Modes

This application holds two game modes: singleplayer and multiplayer, depending on whether a user is performing an inspection by himself or in a team. These modes have been called Challenge and Team.

3.3.1 Challenge

This game mode represents a simple review made by a single user, in which the player does an individual attempt in the game page and after that is finished the results are immediately available, placing the player in the ranking according to the attempt's rating.

3.3.2 Team

In this game mode, players join the chosen game's team lobby, and either start a team or join one that already exists. When the team is ready, the captain will be chosen or automatically assigned. From that point, each player is automatically redirected to the game page where they will start their individual attempts. Those who finish faster will be redirected to a waiting page, where they will wait for everyone in the team to finish. After they are all done with the individual part, they are redirected to the meeting page, where the range of actions depends on whether the user is a team captain or a regular team member. The team members can vote up or down on a defect depending on whether they agree the defect is a correct guess or a wrong guess, while the captain has the responsibility to choose the defects that will be rated and remove from the selection those which the team mates agree that are not to be part of their submission. The defects subject to the team decision in this stage are every defect marked by the players in the team. When the meeting process ends, all the players in the team are redirected to the results page in order to check the rating of the team's performance.

3.4 Rating Calculation and Badges

3.4.1 Rating Calculation

The rating system in this application uses a formula based on the similarity of defects picked by the player or the team and the time spent on game phase. The result is represented as a percentage relatively to the solution.

Each defect comparison has a maximum of 2. This means the solution's score will be the double of the number of defects in the solution plus the maximum time score, which depends on the time spent but also the maximum score on the defect comparison. The value of 2 was chosen

arbitrarily, as any set of values that would maintain each component's weight in the classification would be acceptable.

Starting by the defect comparison, an array with the solution defects is checked item by item. Each item is compared to each guess through a Jaccard index formula. The Jaccard index measures the similarity between two strings. In this case it was adapted to the application, since the program assigns an identifier to every character in the code, so instead of calculating the common characters and the union of strings, like it usually needs to, it deduces that from the start and ending point of each string.

$$Jaccard\ Index\ (a,b) \quad = \quad \frac{Common\ (a,b)}{Union\ (a,b)} \tag{3.1}$$

The best match, if there was at least a comparison with a Jaccard index above zero, the solution defect and the guess defect are then compared for type and severity, comparisons that can take one of two values, according to whether the comparison is verified. Since the Jaccard index can take values from 0 to 1 and the comparisons of type and severity are worth either 0 or 0.5, according to what was already stated regarding the maximum of the defect comparison formula being 2.

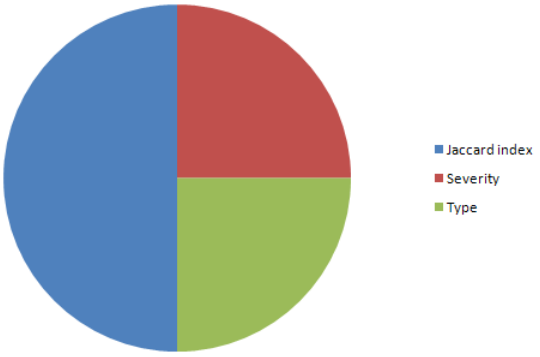


Figure 3.1: Weight of the components of the defect match formula

For each match, if the comparison score is 2, it's counted as correct, otherwise it's defined as an incomplete guess. If the solution defect found no match (Jaccard index equal to zero), it's counted as miss, since it means the player failed to find that defect.

After this verification, the defects the player guessed that weren't matched to any of those in the solution are counted as wrong guesses. As of now, wrong guesses are not being taken into the rating equation as penalties as its need is not yet verified.

Finally, the maximum value of the time component of the rating is calculated as being 25% of the maximum defect matching score. This is meant for the value of the time component to not be a fix value, which could cause its value to be too small for exercises with a relatively large amount of defects to find. Furthermore, as the description suggests, it will depend on the progression of time.

So, if the attempt ends before the time spent reaches 50% of the time limit, the time component will be maximum. However, after that mark, it will decrease in a linear way until it reaches zero, right when the time spent is equal to the time limit.

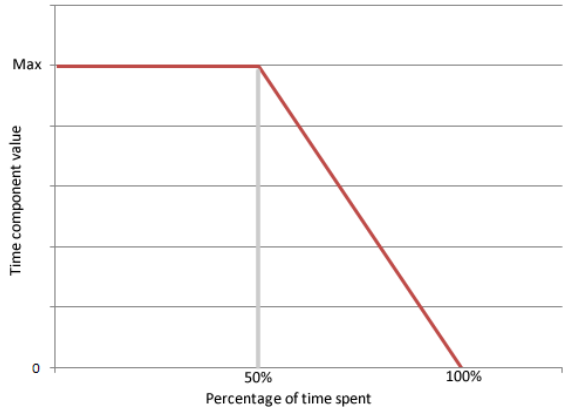


Figure 3.2: Evolution of the time component of the rating formula throughout the game time

The attempt's score is then calculated, like stated, by percentage, comparing the points accumulated in the attempt to the points of the solution.

All these metrics are described at the rankings, although the team total time also has the meeting time added up. Still, that time is not taken into consideration for the rating formula, with the highest time in the team being the selected value to determine the time bonus of the team attempt.

3.4.2 Badges

In the student view, the badges won by the respective student will be displayed, so it allows keeping track of the challenges and milestones achieved and so other players can also view that player's conquests, in an effort to promote competitivity.

According to the final score of each attempt, the player (or team) can be awarded a medal. The medal types are bronze, silver, gold and platinum and their sequence is that of any competition the first three are used with platinum being above gold.

In order for players to be awarded a medal, they need to score at least 50%, in which they will be awarded a bronze medal. To reach the silver medal level, the score must be 75% or above, while a rating of 90% is needed for a gold medal. The platinum medal can only be achieved by scoring a perfect 100% on the attempt.

Earning any medal above bronze will grant you all the medals below. However, only the medal with the highest medal will be shown for a given attempt.

Chapter 4

Validation Experiment

A prototype was developed, based on these specifications, with the intention of testing the concept, by comparing the experiences the subjects have with the application to the main goals.

4.1 Goals and Procedures

The main goals of this experiment, conducted with 4 student volunteers at FEUP, were to verify whether this application is a valid adaptation of the code inspection process and whether it can hold its own as a serious game. In other words, to test whether the steps of the inspection process (individual review and then group review with the results of the first) are being reproduced and whether the players recognize the application as a serious game, a tool they can see being used for code inspection learning.

We can define as secondary goals to investigate the fairness of the rating system and the usability of its tools. That is to say whether the users agree their efforts and performances receive fair ratings and whether the features of the application help reach its goal in a simple and intuitive way.

4.2 Application Adaptation for the Experiment

In order to avoid some problems detected in preliminary testing due to the nature of the resources used, the experiment was done using a basic prototype of the application, focusing on the features that allow the validation of this application's goals. This was made mainly to avoid problems in the communication with the database that would cause issues with the game sequence of actions.

So, the application was split in two modules and hosted in different places according to their needs:

- The main program[1], where the user interacts with the application, hosted at the FEUP server[2].

- An API for database communication[3], as well as a MySQL database, hosted at *OpenShift by Red Hat*[4], an online platform that allows the *NodeJS* and database hosting necessary for this experiment.

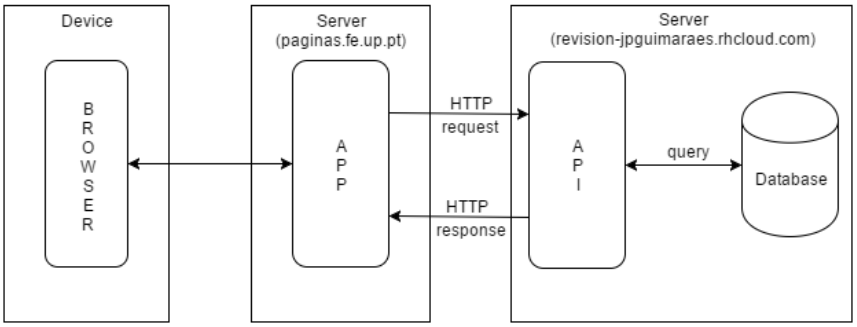


Figure 4.1: Architecture of the experience

The reason the database for this experiment was done in MySQL was that the platform offered better support and managing tools for this database language, which allowed for faster debugging and database testing.

Furthermore, this prototype was designed to avoid any risk of communication failure. Since the server was implemented in a free hosting service, it's natural that there are some limitations in what concerns its operation. It was noticed in the development phase that some HTTP requests would either get a failure response from the server or the respective response would take a longer time than usual to reach the application. There were also some rare occurrences of issues related to the *access-control-allow-origin* header in the server responses. Since these occasional issues didn't seem to be related to any possible problems with the application or the API, it was decided that HTTP requests that would be unnecessary for the purpose of this experiment or those that can be replaced by another action in the experiment without changing its goals would be deactivated and, if needed, changed by a manual action, possible in this scenario since all players are in the same room and team mates can work side to side.

On that note, the requests present in the single player mode, in the automatic page re-directions according to team mate status, in the non-captain part of the meeting page and the updating features of the same page were disabled. The first is justified by the fact that the single player is not needed for this experiment. The requests present on the non-captain view of the meeting page consist of *upvoting* and *downvoting* defects while the updating features of that page consisted in

[1] https://paginas.fe.up.pt/~ei10078/revision/
[2] https://paginas.fe.up.pt/
[3] http://revision-jpguimaraes.rhcloud.com/
[4] https://www.openshift.com/

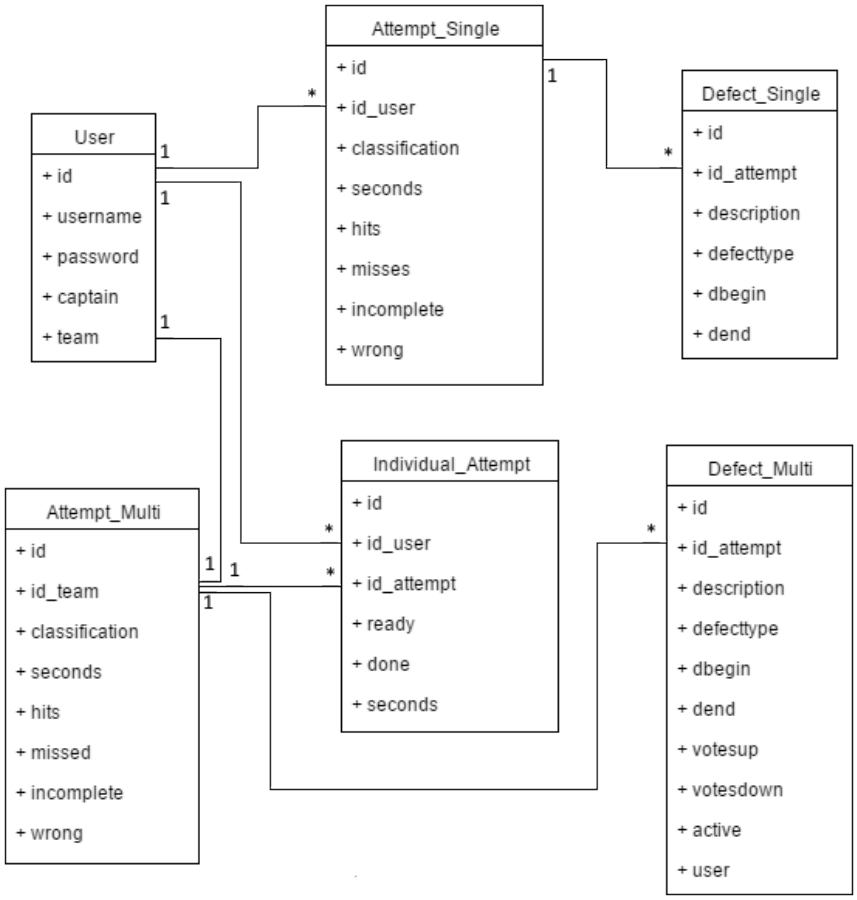


Figure 4.2: Diagram of the modified database for the validation experiment

updating the status of the defects in what concerns *upvotes*, *downvotes* and whether they're selected or unselected (this last check an exclusive of the non-captain view). Keeping these requests would represent a bigger volume of HTTP requests the server would have to deal with, which, as noticed in the development stage, could increase the risk of occurrences of delays or communication failures. So, the solution found was to disable those features and have the team reunite around the captain's device and work together that way. Since the size of each team in the experiment was 2, it was decided that having those minor features wouldn't be worth the risk of failures and delays. So, the main goal was favored, since this decision focus the test of the application's version of the inspection process as a whole. The automatic re-directions after checking the status of team mates were disabled because since the decision of having the meeting part of the inspection be done in the captain's device only was taken, and the players were asked to start the individual part at roughly the same time, they became practically unnecessary in this new situation.

Also for a question of safety against unforeseen trouble and in harmony with the adaptations the application suffered for this experiment, a basic version of the database scheme was developed, to favor the data transactions of this version of the prototype. So, the players start already assigned to a team and their attempts are already registered in the database, although with null values, to be

updated when they finish them. The same happens to the team attempt, as it's already present in the database and only awaits the final information.

4.3 Exercise Selection and Settings

The exercise used for this experiment was an example from ESOF classes, coded in Java (Appendix A). It was decided the time limit would be 30 minutes, similar to the one applied in classes for that same exercise.

Based in the Orthogonal Defect Classification, the chosen types to be available for defect classification were:

- **Documentation**
- **Assignment/Initialization**
- **Checking**
- **Algorithm**
- **Interface**

The severity degrees were defined as:

- **Major** - defects that prevent the program from working or operating as expected
- **Minor** - defects that impact the program in a non-severe way or are irrelevant to its results

The solution to the given exercise is as follows:

Table 4.1: Exercise solution

Type	Severity	First Char Position	Last Char Position
Documentation	Minor	39	274
Assignment/Initialization	Minor	375	388
Checking	Major	919	923
Checking	Major	1144	1148
Assignment/Initialization	Major	1319	1319
Assignment/Initialization	Minor	1400	1400
Assignment/Initialization	Minor	1400	1400
Assignment/Initialization	Minor	1574	1575
Checking	Major	1615	1655
Algorithm	Major	1721	1721
Documentation	Minor	1759	1833

4.4 Subjects

The 4 subjects of this experiment consist in software engineering students, with 3 of them being PhD students and the other being a MSc student. The division and credential attribution of the 4 players followed this plan:

Table 4.2: Teams for the experiment

Team	Player ID	Player Username
1	1	player1
	3	player3
2	2	player2
	4	player4

4.5 Steps

Having that in mind, the experiment occurred as such:

1. The volunteers were asked to make teams of two and decide which one would be the captain
2. They were then asked to access the application and log in with the credentials given to each one, according to their teams and their role
3. A sheet with brief instructions on the process and the defect classification and identification was handed to the players (Appendix B)
4. They were given brief explanation of the inspection process as a whole, site navigation, how to pinpoint defects using the given tool in the game page, the meeting process, and the rating system
5. There was some space for questions from the players in aspects of the application and the procedure they were unclear about.
6. The volunteers were told they could start the challenge
7. Both teams then proceeded to solve the given exercise according to the correct operation of the application
8. As each team finished, they were given a brief explanation of the results page and their options from there, namely consult the ranking table of that exercise for the team category
9. Finally, the players were asked to answer the questions in the feedback form (Appendix C)

4.6 Inputs and Results

The following tables describe which defects each team included in their answers.

Table 4.3: Team 1 attempt

Type	Severity	First Char	Last Char	Detected by
Documentation	Minor	400	414	Player 3
Algorithm	Major	450	622	Player 3
Algorithm	Major	570	601	Player 1
Documentation	Minor	633	697	Player 1
Algorithm	Major	928	1039	Player 3
Documentation	Minor	1054	1066	Player 3
Algorithm	Major	1538	1718	Player 3
Assignment/Initialization	Major	1562	1577	Player 1
Algorithm	Major	2019	2028	Player 3
Algorithm	Major	2032	2047	Player 3
Assignment/Initialization	Major	2278	2291	Player 1
Algorithm	Major	2279	2290	Player 3
Assignment/Initialization	Major	2391	2407	Player 1

Table 4.4: Team 2 attempt

Type	Severity	First Char	Last Char	Detected by
Documentation	Minor	409	419	Player 2
Checking	Minor	972	977	Player 2
Checking	Minor	1321	1327	Player 2
Algorithm	Major	1631	1656	Player 2
Assignment/Initialization	Major	2279	2290	Player 2
Assignment/Initialization	Major	2391	2407	Player 2

After submitting their attempts, each team's efforts are evaluated using the process described in subsection 3.4.1 of this document.

Table 4.5: Result summary

Team	1	2
Defects found in individual attempts	15	7
Maximum time in individual attempts	25m 29s	28m 15s
Defects discarded in meeting step	2	1
Time in meeting step	6m 31s	5m 8s
Correctly signaled defects	0	0
Partially correct signaled defects	2	1
Defects not found	9	10
Wrong guesses	11	5
Total time spent	32m 0s	33m 23s
Final score	10.94%	6.32%

Following this step, each player was asked to fill in a form to give some feedback of the application according to their experience of it. Tables 4.6 and 4.7 detail the players' answers to

that form. The majority of questions were to be answered with a number from 1 to 5 were 1 is the most negative, 5 the most positive and 3 neutral. The last two were open questions.

Table 4.6: Feedback form answers - rate of agreement

	Player 1	**Player 2**	**Player 3**	**Player 4**
How do you feel about using a computer for the inspection process?	4	5	5	3
How would you rate it in comparison to the traditional method?	4	4	4	4
Is the multiplayer mode a good reproduction of the inspection process?	4	4	4	3
How do you rate the evaluation and feedback method when compared to the traditional method (eg: having the solutions told by a teacher)?	3	5	4	4
Do you think the rating system/process is fair?	2	3	5	3
Do you think your performance deserved a higher rating?	3	4	5	4
Would you repeat the experience or use the application to practice?	2	3	4	4

Table 4.7: Feedback form answers - open questions

	Did you find any difficulties?	**Do you suggest any improvement?**
Player 1	Yes, it's not intuitive to use this tool	-
Player 2	No	In the results page, it should be possible to check the type of each defect
Player 3	No	-
Player 4	This was first experience but was good. There was no such difficulties i found which make the tool complex to use	Use colors for code text like in IDEs (eg.: eclipse). brief explanation of program before exercise, so players understand the logic when looking for errors

4.7 Analysis

The experiment occurred with no setbacks, with the application as a whole operating correctly and the data being correctly stored and presented. Although the team results weren't that much appealing, the answers to the feedback form, the time both teams spent on the challenge and the number of defects can both explain the low result and present some overall positive feedback.

Starting with the team game results, the overall values were low, although that can be explained by lack of practice using the tool and being a new experience for the volunteers of this experiment. It spite of that, both team's performances were relatively good, as they managed to go through the procedures with no issues and input a fair amount of defect picks. There's also the specific case of

player 4, who wasn't very familiar with the Java language. Also, there were two unseen mistakes at the code (two cases of words missing a space between them) that weren't intended to be, and were detected by the players. Since there was no penalty for wrong guesses, the impact of this situation was minimum.

Some wrong guesses can also be explained that in some cases, defect classification can be a bit subjective. This should be a point to be improved, as it means there needs to be a better explanation of the defect types, perhaps with examples or via a tutorial or a defect pinpointing rule book, so players can learn faster how to identify defects in the application correctly. The fact that the only defects where the teams got points were incomplete guesses proves the need of such improvements.

The time spent by both teams suggests the time limit was well defined, since it showed there was enough time to review and pinpoint the correct defects, since Team 1 picked 15 excerpts of code. Team 2 was more careful and picked 7 defects in individual game phase, which is not far from the correct 11.

The time spent in the meeting stage, however, was a little more than a third of total time, which suggests that, for exercises of this size, it's a comfortable limit. The fact that there were defects discarded at this point shows the utility of this feature.

There is consensus among the players that using computerized reviews is a better option than traditional reviews and the team inspection mode in this application is a good implementation of the inspection process. Although the majority of the players agreed that the evaluation and feedback methods of the application better than the traditional process (Player 1 thinks the methods of both sides are equally good overall), they seem to agree it needs improvement, although Player 3 thinks it's already fair enough. Since the majority considers their final scores should be higher, this should definitely be a point to improve.

Half of the players admit they would repeat this experience or use the application for practice purposes, which is a good sign the application can meet its goals. Player 2 remained neutral in what regards this subject and Player 1 answered with a 2, which, although representing the player wouldn't repeat it, allied with the rest of the answers the player gave, approving the computerized reviews and the multiplayer method implemented, may indicate the problem, in this player's view, is not in the concept of the application. Player 1's feedback states the application isn't intuitive, but doesn't offer any suggestions to improve it, which can be a sign that the application just needs better explanation in certain aspects. This is consistent with Player 4's suggestion, where he states there should be an explanation of the logic of the code before starting the exercise. Furthermore, Player 1 contributed with 5 of the 13 defect guesses Team 1 submitted as their final answer, which may suggest as well the problem is not in that area.

The remaining suggestions are also useful and should also be integrated in future developments, since using code representations that are similar to the coding and software developing environments helps the player feel more comfortable during the review and something indicating the type for each defect is clearly missing in the exercise results page.

Overall, this was a positive experiment since it demonstrated the application meets its goals and there's room to improve.

Chapter 5

Implementation

In this chapter is described how the designed solution can be implemented.

5.1 Architecture and Technical Specifications

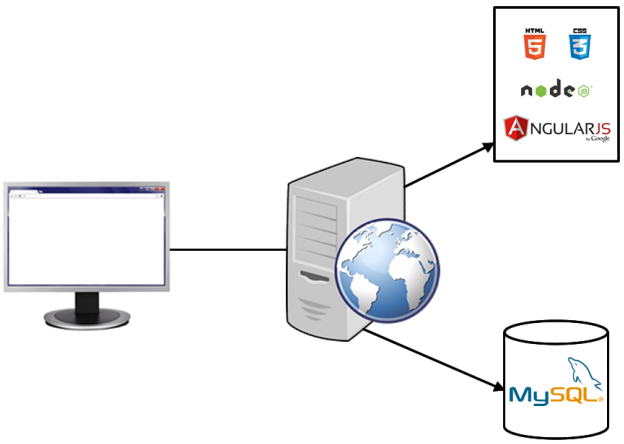


Figure 5.1: Physical description of the system

The proposed solution has two nodes: a server and a client device. The first is a simple website-hosting server while the second can be any device with a web browser compatible with the presented technologies. Figure 5.1 is a possible representation of this layout. The server has a database and two modules, which are the user interface and the database interface.

These elements are described as following:

- The database uses the MySQL language and holds the challenges' data, as well as other necessary information, such as user profiles.

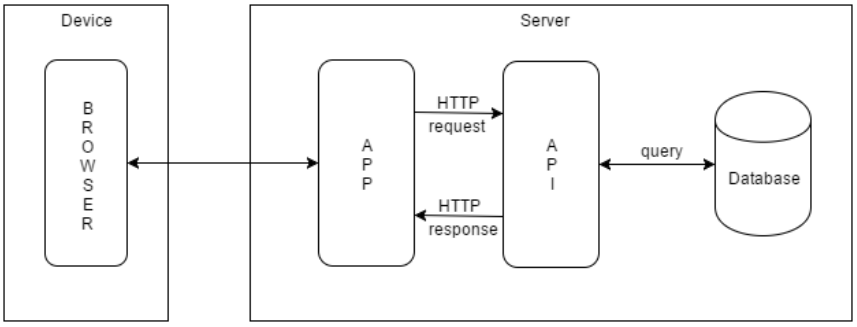


Figure 5.2: Architecture

- The user interface module uses the advantages of HTML5, CSS3 and AngularJS, which allow the execution of all the features related to the execution of the application in a browser. The AngularJS framework[1] is used, since it simplifies the process of connecting all the technologies due to the language's nature. So, the application can be accessed by the most devices possible, since it's accessed via browser, which most devices have nowadays. This framework follows the MVC pattern.

- For the features dealing with database information, a NoseJS module was implemented, holding all the necessary functions, connecting the user interface to the database via an API that handles HTTP requests. The reason this is an independent module is so that the application has a better organization and defined modules working, which will make possible changes simpler. However, it can be hosted in the same location as the rest of the application, as long as its supported.

The technologies were selected based on the reach of their features and the needs of the application, namely web support and compatibility with most devices as possible.

5.2 Database Design

As stated in the previous section, the database's goal is to hold the exercises' data and the users' information. With that in mind, the database was designed as described in Figure 5.3.

Since users are divided in two categories - students and teachers - with different actions inside the application and different ways to access and treat information, their separation has to be reflected in the database. That separation consists in a field in the users' table, which will relate to a table holding the user types, allowing not only the access to the restrict actions of each user group, but also allowing a user to be in multiple groups, in this case, to be a student and a teacher.

Now, in what concerns exercises, it should hold some form of connection to the teacher that posts it. Since only one teacher posts an exercise, that connection can be reduced to a field in

[1] https://angularjs.org/

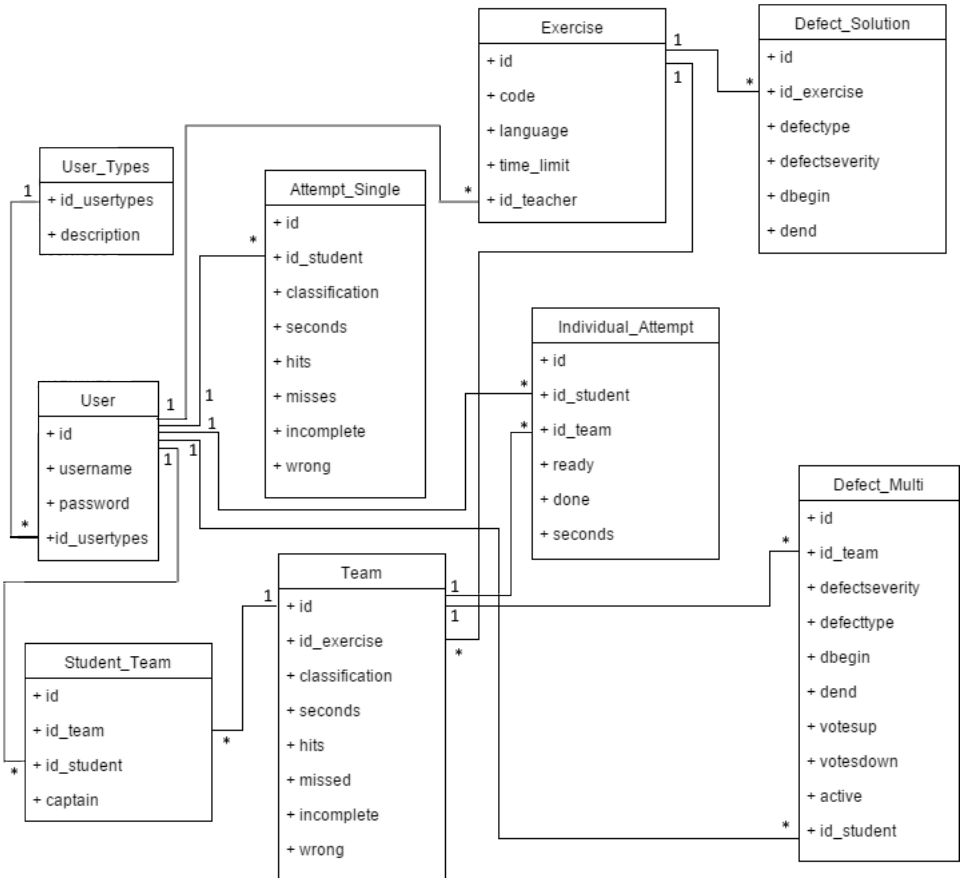


Figure 5.3: Database diagram

the exercise information table identifying the teacher that submitted the exercise. However, the solution defects are not a definite set, varying in number from exercise to exercise. In that way, they have to be stored in a table of their own, with a field indicating the identifier of the exercise. It also must hold the code, its language and the time limit.

As for the student part, it must hold connections to exercises in two ways: through a solution attempt and through a team. In the first, there is just an table with all the attempt information connecting the student and the exercise. As for the team mode database section, it's slightly more complex. The player must be connected to a team and since the player can be in various teams and a team will have more than one player, there needs to be an intermediary table. That intermediary table's id will be the way the individual attempts will be identified, so that the team information can be connected to the individual attempts and gather the individual information needed to produce the team results. The individual attempt has fields described as *ready* and *done*, meaning the player is ready to start its individual attempt and that the attempt is finished, respectively.

Defects in single player attempts are stored in a different table than those related to team games, as they have slightly different data storage needs that wouldn't justify applying the same principle that was applied to the users in this database. The defects of the solutions (already described before) are stored in a different table than the other two types for the same reason. In the

particular case of the defects in the multiplayer mode, it holds 3 more fields so that the defect is ready to answer to the meeting page demands. So, it holds a field for votes in favor, votes against and to signal whether the defect is active or not.

In general, defects hold fields for its severity and type. In this case, in what concerns severity, a boolean value is used, with *false* (or 0) representing *minor* and *true* (or 1) representing *major*. The type field (defectype in the diagram) is a string representing that contains the defect type.

The results of the attempts will be stored in a similar manner, although in different tables for the different game modes. The information stored will be the defects that were found, the defects that were missed, the defects that were found but in an incomplete manner and the wrong guesses, as well as the attempt's rating, using their respective fields.

5.3 Website Design

Since the application follows a MVC pattern, it will be composed of multiple views. Such views differentiate from each other both visually, through the available features and the paths that can be followed from each of that view.

The way the views are organized and can be navigated through is described in Figure 5.4.

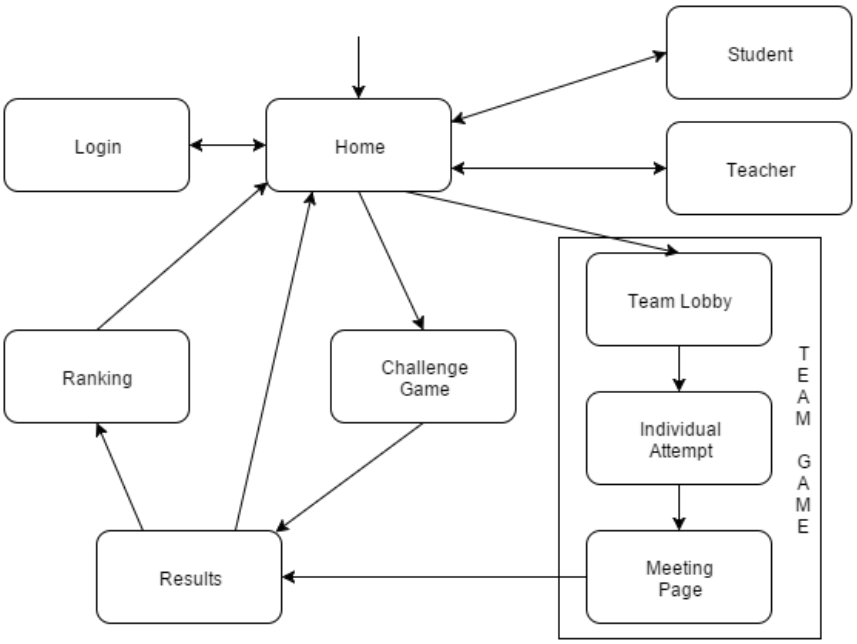


Figure 5.4: Architecture

Next it will be described the main views of this application, some of which were implemented for the validation experiment (described in chapter refchap:valexp).

5.3.1 Login view

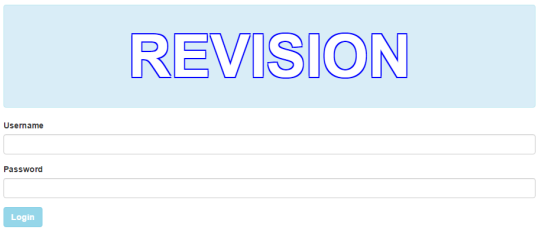


Figure 5.5: Login view

This view welcomes users without a logged in session, no matter what page they try to access. It consists on a form where users insert their usernames and passwords in order to gain access to the application, if their input matches an user registered in the database.

The login and logout feature utilizes local storage of the browser to save the application's session information for the current user. In the case of the login, the credentials inserted are used as arguments in a query to the database to check if there's a register with those credentials. In the affirmative case, it returns the user's identifier, to be stored in local storage and dealt with as previously described.

5.3.2 Home view

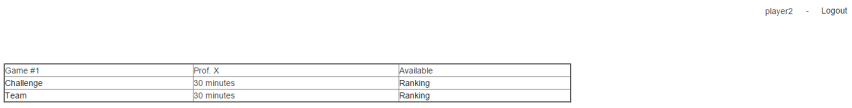


Figure 5.6: Home view

This is the main view for logged users. It lists the available exercises and their information, as well as links for the player's page and to logout of the application.

The exercise list is comprised a series of tables that succeed vertically, in which each table corresponds to an exercise. Each table will have 2 or 3 rows of information, depending on whether both game modes will be available for that exercised. Each game mode row lists the game mode

it corresponds to, the time limit and a link to its respective ranking. The first row of the table describes the game number, the teacher that posted it and the availability of the exercise.

5.3.3 Student view

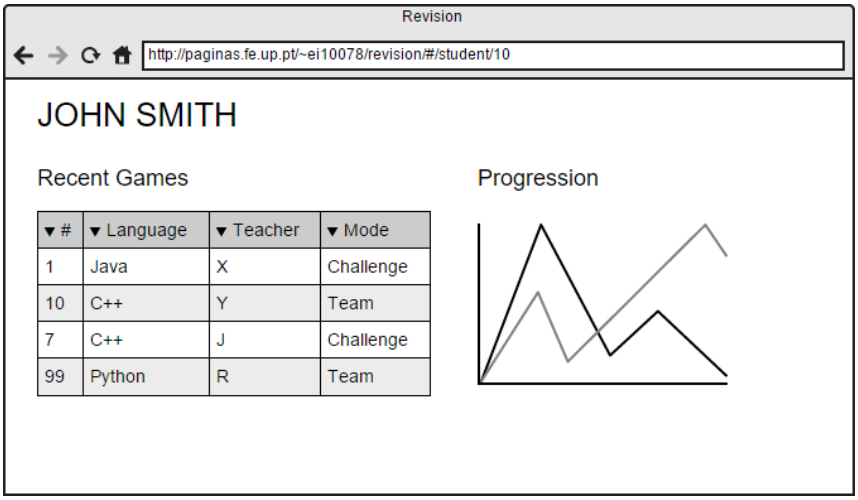


Figure 5.7: Student view prototype

The information related to a specific student and the attempts that student made in any exercise are displayed here. It holds some visual information on the player's evolution as well as the badges won.

5.3.4 Teacher view

This page lists the exercises posted by the teacher, as well as links to the respective games and rankings.

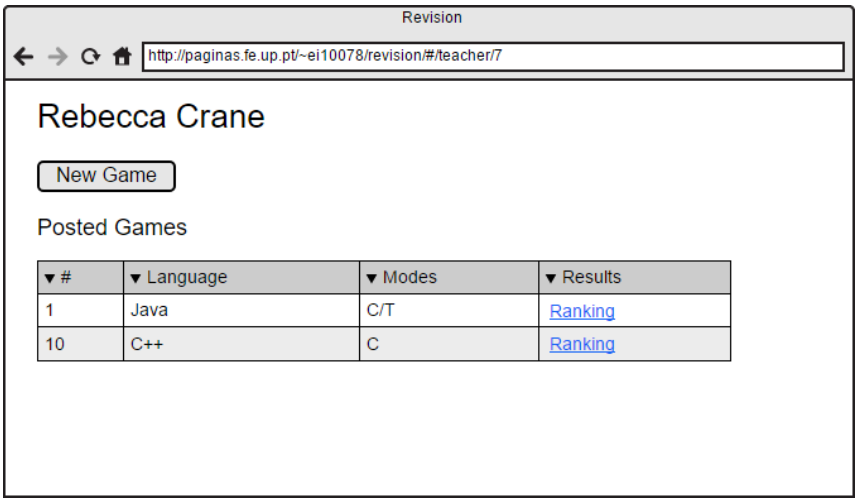


Figure 5.8: Teacher view prototype

5.3.5 Game view

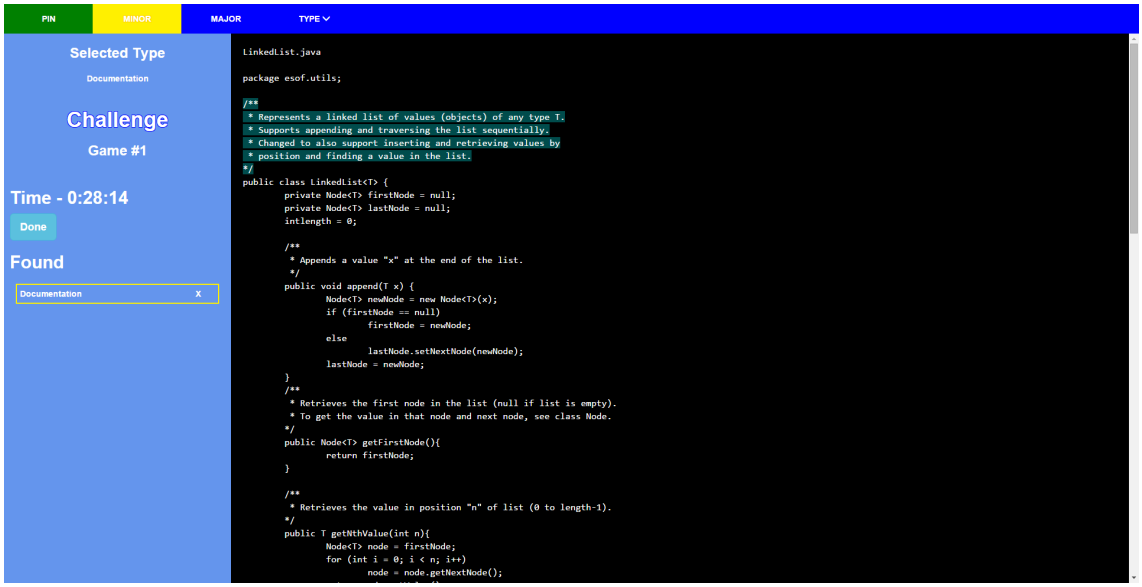


Figure 5.9: Game view

In this view, the revision process of the inspection takes place. It's a common screen for both game modes and it holds all the features related to marking defects.

5.3.6 Meeting view

This view is similar to the game view in layout, but different in its purpose. Instead of marking the defects, it's the job of the team to pick or unpick the already marked defects in order to choose a set to be submitted as a solution attempt to the exercise. It holds a difference in the available

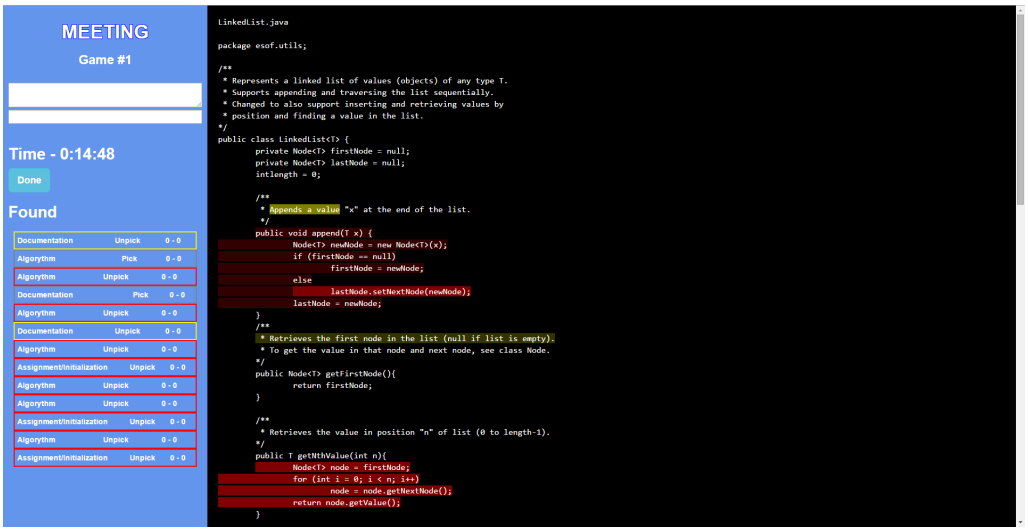


Figure 5.10: Meeting page - captain's view

actions, according to the role of the player. The regular team members will be able to vote up and vote down defects and check their status, which will be regularly updated, while team captain will be able to change the selection status and check the votes in every defect.

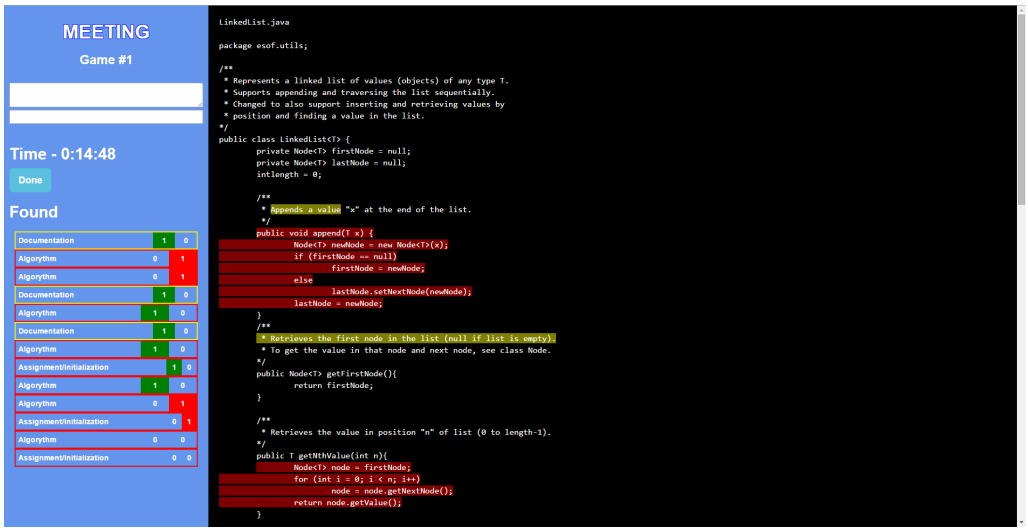


Figure 5.11: Meeting page - othet team member's view

5.3.7 Results view

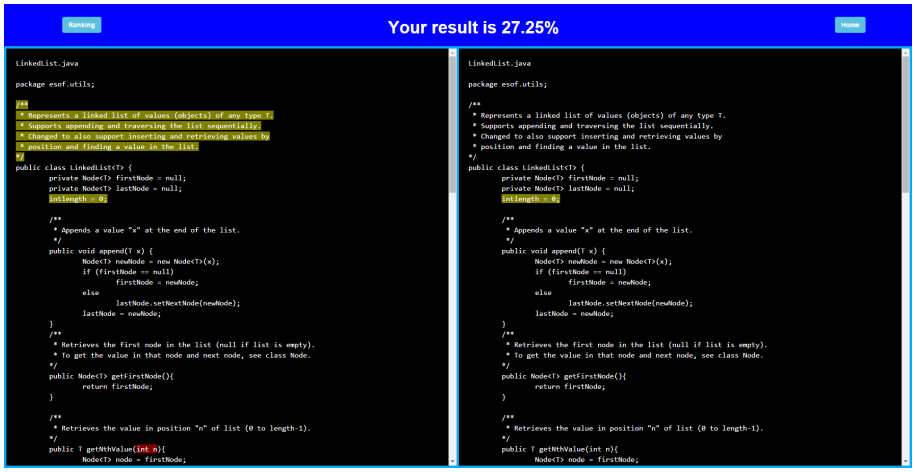


Figure 5.12: Result view

This is another view that is common to both game modes, as it consists in the screen that shows the rating of the solution attempt and a visual comparison to the solution of the respective exercise.

5.3.8 Challenge ranking view

View that lists the completed attempts for a given exercise, in decreasing order of rating. Along with the rating, it also lists the main metrics of the respective attempts, which consist on the time

spent and the defects found, partially found, missed and the number of wrong guesses.

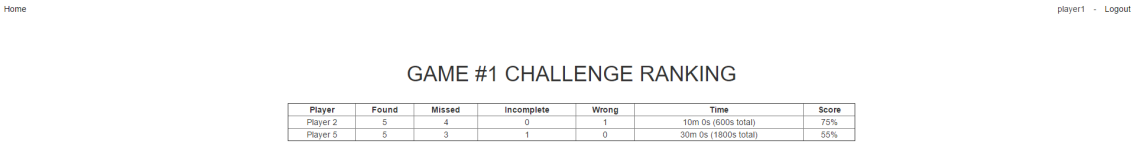


Figure 5.13: Challenge ranking view

5.3.9 Team ranking view

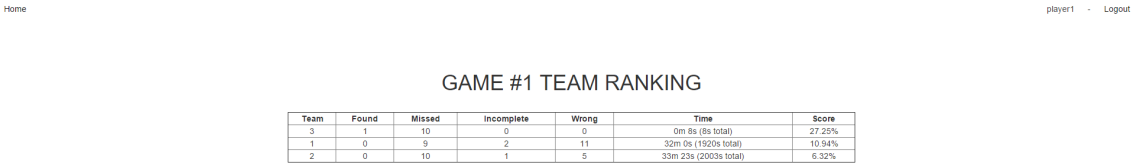


Figure 5.14: Team ranking view

This view is similar to the Challenge ranking view, with the exception that instead of the solution attempt being connected to a player, it's connected to a team identifier, which refers to the players in that team.

5.3.10 Team lobby view

This is the first view players interact with after clicking to start a team game. It's in this page that players either join or start a team to attempt a solution to a team game.

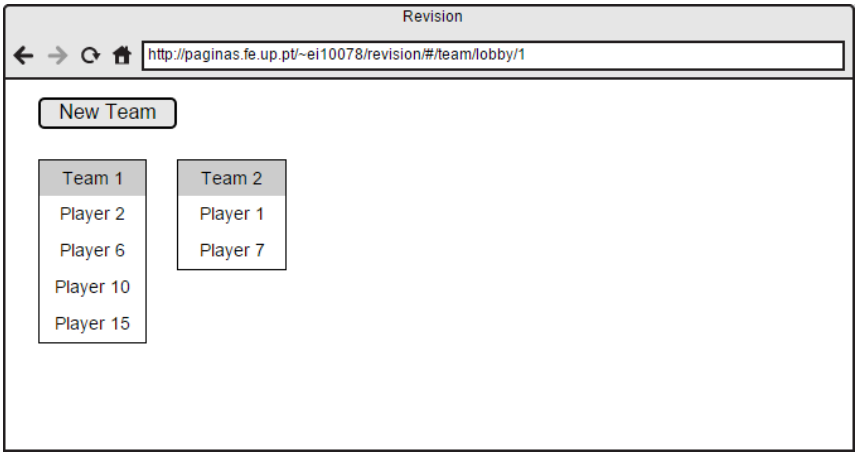


Figure 5.15: Team lobby view prototype

Chapter 6

Conclusions and Future Work

As stated previously, the main goals are to apply *gamification* techniques to turn code revision and inspection learning into a more appealing, effective and efficient process and to contain support for revisions in group in a web based environment. This chapter will describe whether and how the goals for this project were met and describe how can this application evolve in the future.

6.1 Goal Satisfaction

The project followed what was planned, although some difficulties forced a delay on the date of the validation experiment. However, this didn't stop the application prototype to be developed for the validation experiment, in which the processes related to the main goals were tested.

Starting with the first main goal, it was not only possible to turn the inspection process into a game via some *gamification* concepts that were applied throughout the process, but a reproduction of a team inspection was also successfully implemented. The feedback was overall positive and showed that there's a good possibility for this application to be applied for its intended use and there's room to improve. The team inspection mode was seen as a good reproduction of the traditional inspection mode by the players and even seen as an improvement, with computerized revisions having the testers' approval.

As for the web based environment goal, the application was developed using web-supported languages and was successfully implemented in servers available online for the experiment validation. Although needing improvements (which will be covered in the next section), it can be said that the application met this goal as well. The application can be used in any device with a browser that supports javascript and doesn't hold any configuration to block HTTP requests.

In what concerns those goals, it can be said that this project was a success. Not only the main goals were met but the overall feedback from the players that participated in the validation experiment was overall positive and encouraging of the application's future development.

6.2 Future Work

After the analysis of the validation experiment and the feedback from the players that tested the application, it can be concluded that although the main goals were met, there are are various points from which the application can evolve.

As a start, the preparation for the experiment revealed that there has to be a bigger concern to the physical structures. In other words, the optimal scenario for the install of the application would be to have all the components in the same server, in order to reduce to the minimum the time spent in HTTP requests. The application was developed (although it was dimmed not necessary for the validation experiment and therefore, was not tested) having in mind the need for synchronization, so it requires adjustments, if necessary.

There are some features and game aspects that although designed weren't in the development stage and they weren't subject to further investigation. Features like the progression graphs in the student view, the exercise posting by teachers, the team lobby and the difficulty levels fit in that category. Despite their operation and appearance were described in the proper section, it's always a point of focus in future work as it is a tool of feedback to the user and promotes competition, which is an implicit goal of the application, since it's a game.

Finally, the information extracted from the feedback as to be taken into consideration, as it is of most importance. In that way, one of the main focus in future work must be the rating system, since the majority of the players felt it wasn't fair enough. The current rating formula was not seen as completely bad by the players, which may indicate that a few adjustments could make it fair enough to be utilized or, if the difficulty levels route is taken, it can be the rating system of one of the higher levels of difficulty. However, the metrics chosen to the be the formula's arguments appear to be best way to determine the player's performance in the game.

In the suggestion section of the feedback form was the proposition to, in some form, in the results page, allow the each defect's type to be shown. Since this also contributes to improve the feedback the player receives for the attempt at solving an exercise, it would also be an important addition to the set of features of this application. One way to solve this would be to present the defect type via a pop up that would appear near the cursor as the player hovers a defect.

Also in the suggestion part of the feedback, a player suggested that the game view layout should be changed in order to look like the coding section of popular IDEs and that, before any exercise, the program's logic should be explained to the player, in order to facilitate defect spotting. The first suggestion this player made can be helpful in the way it may help the player feel more comfortable analysing the code, since it would be similar to something that player may be familiar to, if the player ha coded before. The latter suggestion can be seen as part of the helping features of a game, since part of the challenge can be understanding what the excerpt of code does. However, like stated, it can be a component in the supporting features of a game, available to be used as a form of help to the player.

References

[Alv13] Edgar Alves. *Jogos Sérios para Ensino de Engenharia de Software*. Faculdade de Engenharia da Universidade do Porto, 2013.

[Bar96] Richard Bartle. Hearts, Clubs, Diamonds, Spades: Pplayers Who Suit Muds. http://gamedevelopment.tutsplus.com/articles/bartles-taxonomy-of-player-types-and-why-it-doesnt-apply-to-everything--gamedev-4173, 1996.

[Bur02] Ilene Burnstein. *Practical Software Testing*. Springer, 2002.

[Coe] António Coelho. Lecture on Serious Games. Faculdade de Engenharia da Universidade do Porto.

[Coe15] António Coelho. Lecture on Gamification. Mestrado em Multimédia - Faculdade de Engenharia da Universidade do Porto, 2015.

[Cre] Fog Creek. Code Review Checklist. http://blog.fogcreek.com/increase-defect-detection-with-our-code-review-checklist-example/.

[HLZ] Robin Hunicke, Marc LeBlanc, and Robert Zubek. *MDA: A Formal Approach to Game Design and Game Research*.

[Hui38] Johan Huizinga. *Homo Ludens*. 1938.

[Laz04] Nicole Lazzaro. *Why We Play Games: Four Keys to More Emotion Without Story*. XEODesign, Inc, 2004.

[Per14] José Pereira. *Jogos Sérios para Ensino de Engenharia de Software*. Faculdade de Engenharia da Universidade do Porto, 2014.

[Pi] Psych-it. Cognitive evaluation theory. http://www.psych-it.com.au/Psychlopedia/article.asp?id=439.

[US14] Vladimir Uskov and Bhuvana Sekar. *Gamification of Software Engineering Curriculum*. Bradley University, 2014.

[UW10] Mary Ulicsak and Martha Wright. *Games in Education: Serious Games*. Futurelab, 2010.

[Wer13] Kevin Werbach. 6 Steps to Effective Gamification | with Kevin Werbach [Transcript]. http://www.engagingleader.com/6-steps-to-effective-gamification-transcript/, May 2013.

[Wie01] Karl E. Wiegers. *Peer Reviews in Software: A Practical Guide*, 2001.

REFERENCES

Appendix A

Exercise Code

```
1  LinkedList.java
2
3  package esof.utils;
4
5  /**
6   * Represents a linked list of values (objects) of any type T.
7   * Supports appending and traversing the list sequentially.
8   * Changed to also support inserting and retrieving values by
9   * position and finding a value in the list.
10  */
11 public class LinkedList<T> {
12   private Node<T> firstNode = null;
13   private Node<T> lastNode = null;
14   intlength = 0;
15
16   /**
17    * Appends a value "x" at the end of the list.
18    */
19   public void append(T x) {
20     Node<T> newNode = new Node<T>(x);
21     if (firstNode == null)
22       firstNode = newNode;
23     else
24       lastNode.setNextNode(newNode);
25     lastNode = newNode;
26   }
27   /**
28    * Retrieves the first node in the list (null if list is empty).
29    * To get the value in that node and next node, see class Node.
30   */
31   public Node<T> getFirstNode(){
32     return firstNode;
33   }
```

```
34
35    /**
36     * Retrieves the value in position "n" of list (0 to length-1).
37     */
38    public T getNthValue(int n){
39      Node<T> node = firstNode;
40      for (int i = 0; i < n; i++)
41        node = node.getNextNode();
42      return node.getValue();
43    }
44
45    /**
46     * Inserts value "x" in position "n" of the list (0 to length).
47     */
48    public void insert(T x, int n){
49      Node<T> newNode = new Node<T>(x);
50      if (n == 0) {
51        newNode.setNextNode(firstNode);
52        firstNode = newNode;
53      } else {
54        Node<T> prev = firstNode;
55        for (int i = 0; i < n; i++)
56          prev = prev.getNextNode();
57        prev.setNextNode(newNode);
58      }
59    }
60
61    /**
62     * Retrieves the position of the first occurrence of value "x"
63     * in the list (between 0 and length), or -1 if not found.
64     */
65    public int find(T x){
66      int index = -1;
67      Node<T> node = firstNode;
68      while (node != null && node.getValue().equals(x)) {
69        node = node.getNextNode();
70        index++;
71      } return index;
72    }
73  }
74
75  Node.java
76
77  package esof.utils;
78
79  /**
80   * Represents a node in a linked list of values (objects) of type T.
81   */
82  public class Node<T> {
```

```
83    private T value;
84    private Node<T> nextNode;
85
86    /**
87     * Creates a new node containing a value "x".
88     * Should only be called from LinkedList.
89     */
90    Node(T x) {
91      value = x;
92      nextNode = null;
93    }
94
95    /**
96     * Sets the next node.
97     * Should only be called from LinkedList.
98     */
99    void setNextNode(Node<T> nextNode) {
100     this.nextNode = nextNode;
101   }
102
103   /**
104    * Retrieves the value stored in this node.
105    */
106   public T getValue(){
107     returnvalue;
108   }
109
110   /**
111    * Retrieves the next node (null if there is none).
112    */
113   public Node<T> getNextNode(){
114     returnnextNode;
115   }
116 }
```

Exercise Code

Appendix B

Experiment Instruction Sheet

Instructions

1. **Login**
2. **Select "Team" in the game menu**
3. In the waiting page, if you're not automatically redirected, remove the /wait/ portion of the URL
4. **Play the game**
   - Pick defects:
     (1) Select the code you want to pick
     (2) Chose the severity of the defect (Minor/Major)
     (3) Chose the type from the list
     (4) Click "Pin"
   - Highlight and scroll to defects you find:
     - simply by clicking on its description
   - Unpick defects:
     - by clicking the "x" of the respective defect in the list
5. **Press "Done" when you finish**
6. In the waiting page, if you're not automatically redirected, remove the /wait/ portion of the URL. *In this particular case, wait for your teammate*
7. **Review in group** – this part can be done in a single computer (the captain's)
   - Captain

  - has to choose which defects are in the final answer, by unpicking the unwanted ones. The unpicked defects will turn gray and will not be part of the rating process
- Other team members
  - can vote up or down the defects to help the captain reach a decision

8. **Press "Done" when you finish**
9. **View results**
10. **Check the ranking by pressing the "Ranking" button or via the proper link in the main page**

Types of defects

This describes the type of defects that can appear in the exercise and the way to pin them.

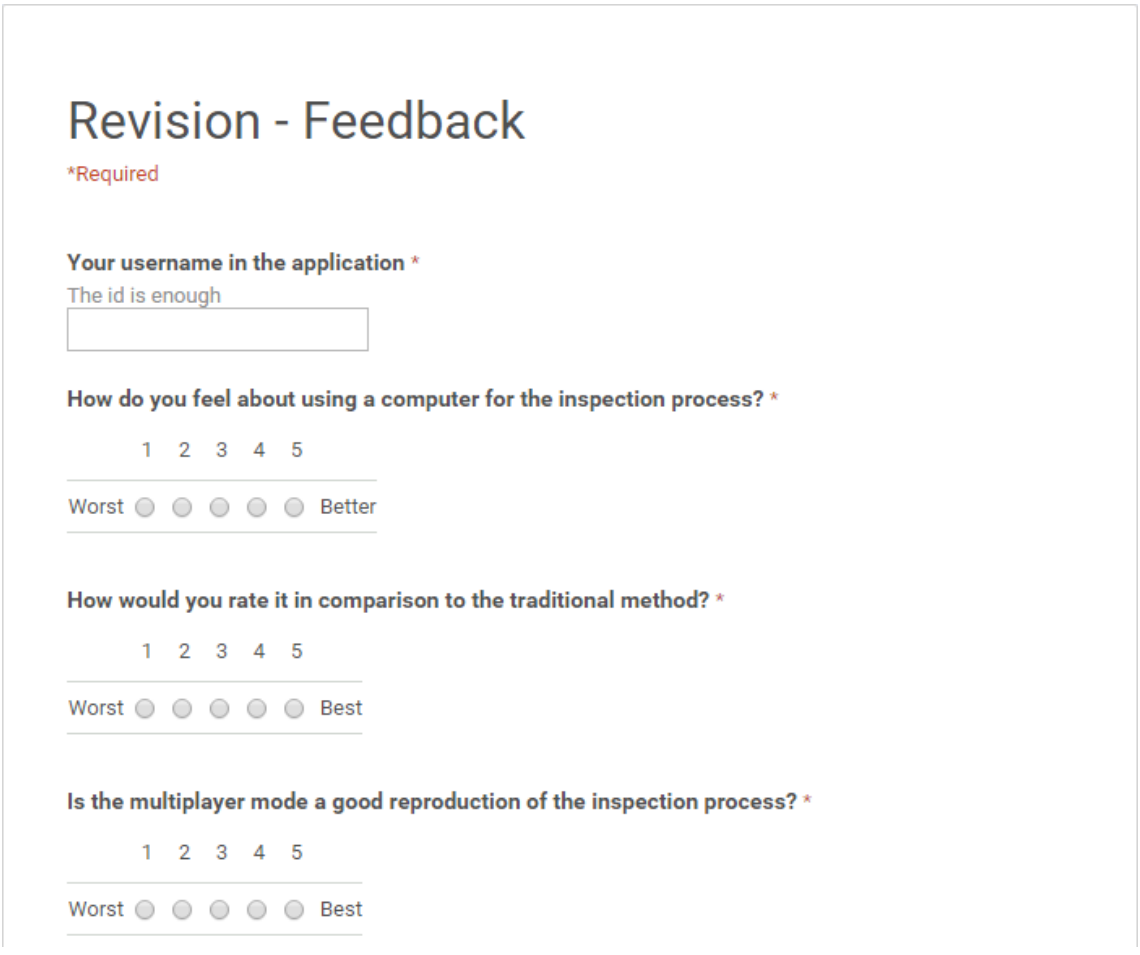
- **Documentation**
  Defects related to documentation of classes, variables and methods, or comments in general.
  - select the whole comment
- **Assignment/Initialization**
  Defects related to variable initialization or assignment.
  - if in the initial value is wrong, select the value
- **Checking**
  When the checking of a condition is not correct
  - select the respective condition
- **Algorithm**
  When the algorithm is not doing what is meant
  - if the issue is in one instruction only, select that instruction
  - if the issue cannot be assigned to a specific instruction, select the closing bracket (}) of that function
- **Interface**
  Issues related to interface procedures
  - select the instruction where that issue is present

Severity

- **Major** – defects that prevent the correct flow of the program or a correct result
- **Minor** – defects that do not interfere in the flow of the program, but its execution isn't optimal

Appendix C

Feedback Form



Revision - Feedback

*Required

Your username in the application *

The id is enough

How do you feel about using a computer for the inspection process? *

1 2 3 4 5

Worst ○ ○ ○ ○ ○ Better

How would you rate it in comparison to the traditional method? *

1 2 3 4 5

Worst ○ ○ ○ ○ ○ Best

Is the multiplayer mode a good reproduction of the inspection process? *

1 2 3 4 5

Worst ○ ○ ○ ○ ○ Best

Figure C.1: Feedback form - part 1

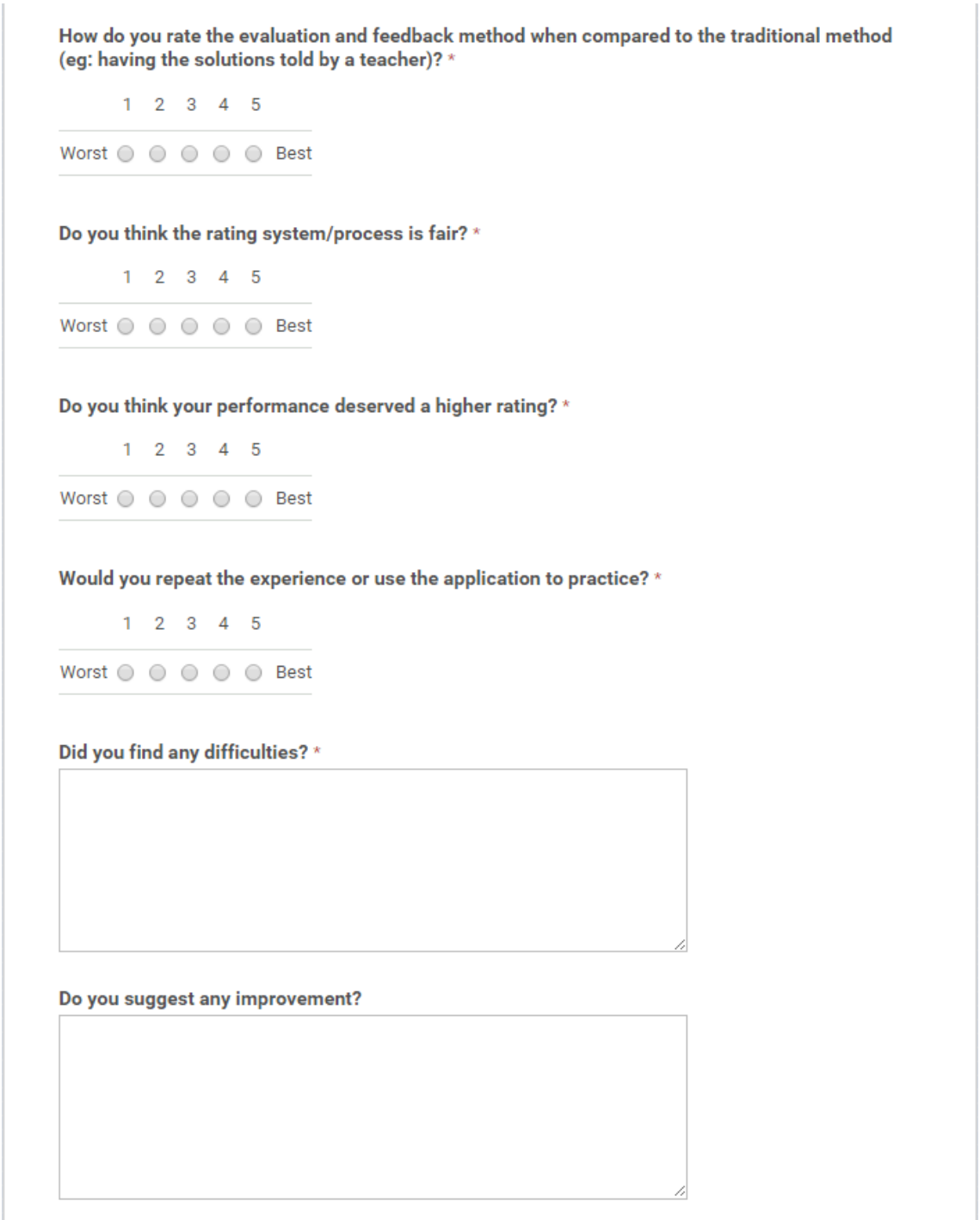


Figure C.2: Feedback form - part 2